Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	63	"allocation site"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/08 17:14
L2	479	"directly called"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/08 17:14
L3	4	"virtually called"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/08 17:15
L4	1	1 and 2	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/08 17:15
L5	19	1 and "run-time"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/08 17:33
L6	0	methd same body same program	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/08 17:34
L7	3367	method same body same program	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/08 17:34
L8	9453	direct same virtual	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/08 17:34
L9	51	7 and 8	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/08 17:34
L10	9	9 and call and graph	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/08 17:34

US-PAT-NO: 6654951

DOCUMENT-IDENTIFIER: US 6654951 B1

TITLE: Removal of unreachable methods in object-oriented applications based on program interface analysis

DATE-ISSUED: November 25, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Bacon; David Francis	New York	NY	N/A	N/A
Laffra; Johannes C.	Yorktown Heights	NY	N/A	N/A
Sweeney; Peter Francis	Spring Valley	NY	N/A	N/A
Tip; Frank	Mount Kisco	NY	N/A	N/A

US-CL-CURRENT: 717/154, 717/151

ABSTRACT:

The present invention analyzes an application A and computes a set reachable methods in A by determining the methods in A that may be called from another reachable method in A, or from within a class library L used by A without analyzing the classes in L. The invention may be used as an optimization to reduce application size by eliminating unreachable methods.

19 Claims, 5 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 5

---------- KWIC ---------

Brief Summary Text - BSTX (5):

Object-oriented programming languages provide a number of features such as classes, inheritance, multiple inheritance, and virtual inheritance. These object-oriented features have several advantages. Most importantly, they enable the creation of class libraries that can be reused in many different applications. Class libraries are typically shipped independently from applications that use them. Libraries are commonly shipped as a combination of the executable (object) code, in combination with the library's interface. In order to use a library, a client application calls functions specified in the library's interface, and includes (links) the library's object code. This model has the advantage that a library has to be shipped and installed only once, even when multiple applications use it. An additional benefit of this approach is that the library's source code need not be exposed. Only the library's external interface needs to be visible.

Brief Summary Text - BSTX (11):

The invention may be used as an optimization to reduce application size by eliminating unreachable methods. In the alternative, the invention may be used as a basis for optimizations that reduce execution time (e.g., by means of call devirtualization), and as a basis for tools for program understanding and debugging.

Detailed Description Text - DETX (8):

The execution of line [3] of the program of FIG. 2 results in the creation of an A object, to which variable i is made to refer. Subsequently, on line [4] method f( ) is called on variable i. Since i points to an object of type A, this method call is dynamically dispatched to method A.f( ). Similarly, the call i.g( ) on line [5] is dynamically dispatched to A.g( ). After executing line [6], variable i refers to a newly allocated B-object. The call i.f( ) on line [7] dynamically dispatches to method B.f( ) because at that point i refers to an object of type B. However, note that the call i.g( ) on line [8] dispatches to A.g( ) because class B does not provide an overriding definition for g( ).

Detailed Description Text - DETX (11):

FIG. 3 shows a Java program that contains classes L, A, B, and C. It will be assumed that L is a library class that is not available for analysis. The program of FIG. 3 has the following characteristics: Interface L specifies two

methods, f( ) and h( ), as well as a constructor method L( ). Class A is derived from L, overrides method h( ) and contains another method g( ) and a constructor method A( ). Method A.g( ) contains a virtual call to method A.h( ). Class B is derived from A, and contains overriding definitions of methods g( ) and h( ), and a constructing method B( ). Class B also contains a method k( ). Method B.h( ) contains a call to method k( ). Class C contains the main( ) method for this application. Method main( ) creates a B-object and assigns it to a variable .alpha. of type A. Then, method f is invoked on .alpha., and subsequently method g( ) is invoked on .alpha..

Detailed Description Text - DETX (12):

Consider the call this.h( ) in library method L.f( ). Since this method is called on an object whose run-time type B, the call will dispatch to method B.h( ). Hence, B.h( ) is a reachable method, even though the application contains no calls to any method h( ) outside the library.

Detailed Description Text - DETX (13):

The method of the present invention determines a set of reachable methods in an application that uses a class library without analysis of the code in the library. The method requires knowledge of the interface to the class library that includes the following: the set of classes (and methods in those classes) that may be subclassed by the application, and the methods in the library classes that may be overridden by applications that use the library. The method of the present invention uses this information to determine the methods in application code that may be invoked by dynamic dispatches in the library's code. The method comprises the following steps: (1) Determining a set of initially reachable methods. This set typically includes the application's main( ) method, and initialization methods for statically scoped or globally scoped variables. (2) Processing a method. This involves the determination of a set of call sites in that method that may be executed, and a set of classes that may be instantiated in that method. (3) Processing a call site. This involves the determination of a set of methods that can be reached from that call site by way of a dynamic dispatch. (4) Determining the set of library methods that are overridden by the application, and that may be executed as a result of a dynamic dispatch in the library's executable code. Steps (2),(3), and 4) are preferably performed repeatedly because a call site found in the course of performing Step (2) may lead to the determination of additional reached methods, and a method that is determined to be reachable in the course of performing Steps (3) or (4) may contain additional call sites, or instantiate additional classes.

Detailed Description Text - DETX (16):

The procedure findReachableMethods( ) (lines [1]-[38]) is the main procedure of the method for determining reachable methods in the presence of library usage. The method is iterative and relies on the following data structures: processedClasses is a set of classes that have been determined to be instantiated, and with respect to which the call sites in set processedSignatures have been processed. Line [3] initializes processedClasses to be the empty set. currentClasses is a set of classes that have been determined to be instantiated, and with respect to which the call sites in the sets currentSignatures and processedSignatures will be processed in the current iteration. Line [4] initializes currentClasses to be the empty set. newClasses is a set of classes that are determined to be instantiated in the current iteration. In the next iteration, the call sites in processedSignatures and currentSignatures will be processed w.r.t. these classes. Line [5] initializes newClasses to be the empty set. processedMethods is a set of methods that have been determined "reachable", and that are fully processed. Line [6] initializes processedMethods to be the empty set. currentMethods is a set of methods that have been determined "reachable", and that will be processed in the current iteration. Line [7] initializes currentMethods to be the set of initially reachable methods. This includes the main method of the application as well as initializer methods for classes. newMethods is a set of methods that are determined "reachable" in the current iteration, and that will be processed in the next iteration. Line [8] initializes newMethods to be the empty set. processedSignatures represents a set of call sites to virtual methods that have been determined as "reachable". The call sites in processedSignatures have been resolved w.r.t. the classes in processedClasses. Line [9] initializes processedSignatures to be the empty set. currentSignatures represents a set of call sites to virtual methods that have been determined as "reachable". The call sites in currentSignatures will be resolved w.r.t. the classes in processedClasses and currentClasses in the current iteration. Line [10] initializes currentSignatures to be the empty set. newSignatures represents a set of call sites to virtual methods that are determined as "reachable" in the current iteration. These call sites will be resolved w.r.t. the classes in processedClasses and currentClasses in the next

iteration. Line [11] initializes newSignatures to be the empty set.

Detailed Description Text - DETX (17):
Turning now to the procedure, lines [12]-[35] perform an iterative process that continues as long as one of the sets currentMethods, currentSignatures, or currentClasses is not empty. In lines [13]-[15], each reached method m in currentMethods is processed by a call to procedure processMethod. In lines [16]-[18], each reached call site d.f( ) in processedSignatures is processed w.r.t. the classes in set currentClasses, by a call to procedure processCallSite. In lines [19]-[21], each reached call site d.f( ) in currentSignatures is processed w.r.t. the classes in set processedClasses by a call to procedure processCallSite. In lines [22]-[24], each reached call site d.f( ) in currentSignatures is processed w.r.t. the classes in set currentClasses by a call to procedure processCallSite. In line [25], the set of overridden library methods w.r.t. the classes in set currentClasses is determined through a call to procedure handleOverriddenLibraryMethods.

Detailed Description Text - DETX (19):
After the iterative process has completed, the set liveMethodDefinitions of reachable methods consists of the set of methods in processedMethods (line [36]). In addition, the signatures of the methods that occur in processedSignatures but not in processedMethods are needed to ensure that method dispatches can be executed correctly (line [37]). For methods in the set liveMethodSignatures, only the signature (and not the body) is required for the execution of the program. For these methods, the body is preferably replaced by one or more statements that satisfy the type-checking requirements of the language (such as a return statement) such that compilation of the program does not result in an error and the size of the program representation is reduced.

Detailed Description Text - DETX (21):
The procedure processMethod (lines [39]-[56]) processes a method m to determine if it contains (additional) reachable call sites, and/or if it instantiates (additional) classes.

Detailed Description Text - DETX (23):
Lines [46]-[50] are concerned with determining reachable direct (i.e., non-virtual) calls to other methods within method m. Specifically, for each reachable direct call to a method A.f( ), it is determined if A is a library class or if A.f( ) already occurs in processedMethods, currentMethods, or newMethods (line [47]). If this is not the case, method A.f( ) is added to newMethods (line [48]).

Detailed Description Text - DETX (24):
Lines [51]-[55] are concerned with determining reachable virtual calls to other methods within method m. Specifically, for each reachable virtual call site to a method B.g( ), it is determined if B is a library class or if B.g( ) already occurs in processedSignatures, currentSignatures, or newSignatures (line [52]). If this is not the case, method B.g( ) is added to newSignatures (line [53]).

Detailed Description Text - DETX (25):
The reachability of a (virtual or direct) call site, or a class instantiation site within a method can be determined using a variety of techniques. For instance, one can simply assume that all call sites that occur in a reached method are always reachable. Alternatively, one may use existing data flow analysis techniques (see e.g., A. V. Aho, R. Sethi, and J. D. Ullman, `Compilers: Principles, Techniques and Tools`, Addison-Wesley, 1986) to determine a more precise approximation of the set of call sites that can reached.

Detailed Description Text - DETX (27):
The procedure processCallSite (lines [57]-[67]) processes a virtual call site A.f( ) with respect to a set of instantiated classes S, and determines a set of (additional) methods that may be reached from that call site.

Detailed Description Text - DETX (35):
It will now be discussed how the method described above determines a set of reachable methods for the program of FIG. 3. For the purposes of this example, it will be assumed that any call site and class instantiation that occurs in the body of a method is reachable (lines [41,46,51]).

Detailed Description Text - DETX (38):
The first iteration of the procedure findReachableMethods proceeds as follows. Since currentMethods is not empty, the body of the while-loop on

lines [12]-[35] is traversed. In the first iteration of the while-loop, C.main( ) is the only method that occurs in currentMethods. Consequently the execution of lines [13]-[15] results in a call to procedure processMethod for method [C.main( )].

Detailed Description Text - DETX (39):
Execution of procedure processMethod for method [C.main( )] proceeds as follows. Method C.main( ) contains an instantiation of class B. Since class B does not occur in any of the sets processedClasses, currentClasses, or newClasses, execution of lines [42]-[44] adds class B to newClasses. Method C.main( ) contains a direct call to the constructor method B.B( ). Since this method does not occur in any of the sets processedMethods, currentMethods, or newMethods, execution of lines [46]-[50] adds method B.B( ) to newMethods. In addition, method C.main( ) contains a virtual call to methods L.f( ) and A.g( ). Method L.f( ) occurs in library class L and is hence not added to any of the worklists described above. Method A.g( ) occurs in an application class and does not occur in any of the sets processedSignatures, currentSignatures, or newSignatures. Therefore, execution of lines [51]-[55] results in the addition of A.g( ) to newSignatures. This completes the processing of all methods in currentMethods.

Detailed Description Text - DETX (41):
Execution of line [25] results in a call to handleOverriddenLibraryMethods with the empty set as an argument. Execution of this procedure does not traverse the loop of lines [70]-[79] because the argument S to method handleOverriddenLibraryMethods is the empty set.

Detailed Description Text - DETX (46):
In procedure processMethod, it is determined that method B.B( ) contains a direct call to constructor A.A( ). Since A is not a library class, and A.A( ) does not yet occur in processedMethods, currendMethods, or newMethods (line [47]), A.A( ) is added to newMethods (line [48]).

Detailed Description Text - DETX (53):
Execution of the loop on lines [13]-[15] results in the invocation of procedure processMethod for all three methods in currentMethods (i.e., A.A( ), B.g( ), and B.h( )). In the course of processing method A.A( ), a direct call to constructor L.L( ) is encountered. Since this method occurs in a library class, it is not added to newMethods. Method B.g( ) does not contain any class instantiation sites, or direct or virtual calls. Hence, none of the variables are affected while processing this method. Processing method B.h( ) leads to the identification of a virtual call to method B.k( ). Hence, B.k( ) is added to newSignatures. Next, the loop on lines [16]-[18] is executed. This results in a single call to procedure processCallSite with the arguments A.g( ) and the empty set. Since the latter argument is the empty set, the main loop of procedure processCallSite is not traversed, and no variables are affected.

Detailed Description Text - DETX (54):
Since currentSignatures is empty, the loops of lines [19]-[21] and lines [22]-[24] are not traversed. Execution of line [25] results in a call to handleOverriddenLibraryMethods with the empty set as an argument. Hence, the main loop of handleOverriddenLibraryMethods is not traversed. Execution of line [26] does not affect processedClasses. Line [27] assigns the empty set to currentClasses. Line [28] assigns the empty set to newClasses. Line [29] adds methods A.A( ), B.g( ) and B.h( ) to processedMethods. Line [30] assigns the empty set to currentMethods. Line [31] assigns the empty set to newMethods. Line [32] does not affect processedSignatures. Line [33] adds B.k( ) to currentSignatures. Line [34] assigns the empty set to newSignatures.

Detailed Description Text - DETX (57):
Next, the loop on lines [16]-[18] is executed. This results in a single call to procedure processCallSite with the arguments A.g( ) and the empty set. Since the latter argument is the empty set, the main loop of procedure processCallSite is not traversed, and no variables are affected.

Detailed Description Text - DETX (58):
Because currentSignatures contains a single element, B.k( ), the body of the loop on lines [19]-[21] is traversed once, and procedure processCallSite is invoked with arguments B.k( ) and B. As a result, the loop on lines [59]-[66] is traversed once, with variable C bound to class B and variable A also bound to class B. Consequently, the condition on line [60] evaluates to true, and lines [61]-[64] are executed. Execution of line [61] determines that a virtual call to B.k( ) on an object of type B resolves to method B.k( ). Since B is not a library class, and B.k( ) does not yet occur in processedMethods, currentMethods, or newMethods (line [62]), method B.k( ) is added to

newMethods.

Detailed Description Text - DETX (59):
Execution resumes at line [22], resulting in a call to method processCallSite with arguments B.k( ) and the empty set. Hence, the main loop on procedure processCallSite is not executed, and no variables are affected.

Detailed Description Text - DETX (60):
Execution of line [25] results in a call to handleOverriddenLibraryMethods with the empty set as an argument. Hence, the main loop of handleOverriddenLibraryMethods is not traversed.

Detailed Description Text - DETX (64):
Execution of the loop on lines [13]-[15] results in the invocation of procedure processMethod for method B.k( ). Method B.k( ) does not contain any class instantiation sites, or direct or virtual calls. Hence, none of the variables are affected while processing this method.

Detailed Description Text - DETX (67):
Execution of line [25] results in a call to handleOverriddenLibraryMethods with the empty set as an argument. Execution of this procedure does not traverse the loop of lines [70]-[79] because the argument S to method handleOverriddenLibraryMethods is the empty set.

Detailed Description Text - DETX (70):
At the end of the fifth iteration of the procedure findReachableMethods, the sets currentClasses, currentMethods, and currentSignatures are all empty, and the loop on lines [12]-[35] is exited, and execution proceeds at line [36]. Line [36] assigns the set of methods in processedMethods (i.e., [C.main( ), B.B( ), A.A( ), B.g( ), B.h( ), B.k( )]) to variable liveMethodDefinitions, indicating that these methods are reachable. Furthermore, line [37] assigns the set of methods that lie outside the intersection of processedMethods and processedSignatures (i.e., [A.g( )]) to variable liveMethodSignatures, indicating that these signatures are reachable, and that these signatures are needed in order to correctly execute virtually dispatched method calls in the program.

Detailed Description Text - DETX (71):
While the description above has relied on the class hierarchy and instantiated classes in order to resolve virtually dispatched method calls, the method can easily be adapted to use only class hierarchy information. In addition, techniques for determining pointer aliases can be used to resolve virtual method calls (See, e.g., H. D. Pande and B. G. Ryder, "Data-flow-based Virtual Function Resolution," Proceedings of the Third International Symposium on Static Analysis (SAS'96), Springer-Verlag Lecture Notes in Computer Science, Volume 1145, pages 238-254, September 1996; and B. Steensgaard, "Points-to Analysis in almost linear time," Proceedings of the 23rd ACM Symposium on Principles of Programming Languages (POPL'97), St. Petersburg, Fla., January 1996, pp. 32-41.

Detailed Description Text - DETX (72):
The technique of the present invention may be used by a tool to reduce the size of a program's representation. A tool accepts as an input a source program and operates on it to generate as an output a more efficient (yet functionally equivalent) representation of the source program. The tool typically includes functionality that parses and performs semantic analysis on the source program to identify the object oriented features of the source program as described herein. A more detailed description of such a tool is described in F. Tip, C. Laffra, P. F. Sweeney, D. Streeter, "Size Matters: Reducing the Size of Java Class File Archives," IBM Technical Report, RC 21321, IBM T. J. Watson Research Center, October 1998, herein incorporated by reference in their entirety. In such a tool, the methodology described above may be used to identify a set of reachable methods of the program. The methods of the source program that are not included in this set may be excluded from the representation of the source program output by the tool, which reduces the size of the program's representation. In addition, the methodology described above may be used to identify a set of methods of the source program whose signature is required for execution of the program yet whose body is not required for the execution of the program. For these methods, the body is preferably replaced by one or more statements that satisfy the type-checking requirements of the language (such as a return statement) such that compilation of the program does not result in an error and the size of the program's representation is reduced.

Detailed Description Text - DETX (74):

In an alternate embodiment, the techniques of the present invention may be used by such a tool to transform virtual calls into direct calls. More specifically, the methodology of the present invention as described above may be applied to an input program that uses methods in a class library to identify a set of reachable methods in the input program. The tool then identifies the virtual function call sites in the input program. For each virtual call to a method, denoted method f( ), the tool determines if the set of reachable methods contains only one method with the same signature as the method f( ). If so, the tool replaces the semantics of the virtual call to method f( ) with a direct call to method f( ), which improves the efficiency of executing the program when the method f( ) is invoked.

Detailed Description Text - DETX (77):
The technique of the present invention may also be integrated into an application for program understanding and debugging. More specifically, the methodology described above may be applied to an input program that uses methods in a class library to identify one or more of the following elements without analysis of the code in the library: i) the methods of the input program that are reachable; ii) the methods of the input program that are not-reachable; iii) the methods of the input program whose signature is required for execution of the program yet whose body is not required for the execution of the program. The identified element(s) are then reported to the user via the display device 38 or other user interface device for program understanding purposes or debugging purposes.

Claims Text - CLTX (3):
3. The method of claim 1, further comprising the steps of: for a given method m in the P, identifying a third set containing call sites reachable within the m; identifying the call sites of the third set as reachable; and excluding call sites of the P that are not reachable from said representation of the P.

Claims Text - CLTX (5):
5. The method of claim 1, further comprising the steps of: for a given call site c in the P, identifying a fifth set containing methods in the P reachable from the c; identifying the methods of the fifth set as reachable; and excluding methods of the P that are not reachable from said representation of the P.

Claims Text - CLTX (6):
6. A method for generating a representation of an object-oriented program which uses an interface of a class library, the method comprising the steps of: identifying a first set containing initially reachable methods in the program; for a given method in the program, identifying a second set containing calls reachable in the given method, and identifying a third set containing classes instantiated in the given method; for a given call in the program, identifying a fourth set containing methods in the program reachable by a dynamic dispatch from the given call; determining a set of class library methods overridden by the program based on the interface, wherein the set of class library methods can be executed as a result of a dynamic dispatch from executable code of the class library; identifying the calls of the second set, the classes of the third set, and the methods of the first, third, and fourth sets, and the library method sets as reachable; and excluding calls, classes, and methods of the program that are not reachable from the representation of the program.

Claims Text - CLTX (9):
9. The method according to claim 6, wherein said step of identifying the second set comprises the step of identifying all calls in the given method as reachable, when the given method is reachable.

Claims Text - CLTX (10):
10. The method according to claim 6, wherein said step of identifying the second set comprises the step of analyzing data flow one of to and from calls in the given method.

Claims Text - CLTX (11):
11. The method according to claim 6, wherein said step of identifying the second set comprises the step of excluding calls in methods corresponding to a library class from the second set.

Claims Text - CLTX (18):
18. The method according to claim 6, further comprising the step of iteratively repeating said step of identifying the second set, the third set, the fourth set, and the fifth set, until all reachable calls, classes, and methods have been identified.

Other Reference Publication - OREF (6):
"Inheritance Graph Assessment Using Metrics", Jean Mayrand et al, IEEE Proceedings of Metrics, Aug. 1996, pp. 54-63.*

US-PAT-NO: 6629123

DOCUMENT-IDENTIFIER: US 6629123 B1

TITLE: Interception of unit creation requests by an automatic distributed partitioning system

DATE-ISSUED: September 30, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Hunt; Galen C.	Bellevue	WA	N/A	N/A

US-CL-CURRENT: 718/106, 717/131 , 719/310

ABSTRACT:

An automatic distributed partitioning system (ADPS) intercepts function calls to unit activation functions that dynamically create application units, such as a component instantiation function. A system service library provides a unit activation function. An application program includes at least one function call to the unit activation function. The ADPS redirects the function call to instrumentation of the ADPS. In one technique, the ADPS uses inline redirection of the function call to the unit activation function.

23 Claims, 18 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 18

---------- KWIC ---------

Detailed Description Text - DETX (2):

The present invention is directed toward automatic partitioning of units of an application and distribution of those units. In the illustrated embodiment of the present invention, an application is partitioned into one or more application units for distribution in a distributed computing environment. The COIGN system is one possible refinement of the illustrated ADPS that automatically partitions and distributes applications designed according to the Component Object Model ("COM") of Microsoft Corporation of Redmond, Wash. Briefly described, the COIGN system includes techniques for identifying COM components, measuring communication between COM components, classifying COM components, measuring network behavior, detecting component location constraints, generating optimal distribution schemes, and distributing COM components during run-time.

Detailed Description Text - DETX (67):

A distribution scheme 50 is the result of applying the environment description set 230 to the application description set 220. The distribution scheme 250 includes a mapping of application units to locations in a distributed computing environment. The units can be classified using static metadata of the units. Alternatively, where run-time profiling was used to dynamically describe the units, the units can be classified according to dynamic behavior. At run-time, units of the application 200 are mapped using the distribution scheme 250 for location on an appropriate computer in the distributed computing environment.

Detailed Description Text - DETX (83):

Dynamic analysis provides insight into an application's run-time behavior. The word "dynamic," as it is used here, refers to the use of run-time analysis as opposed to static analysis to gather data about the application. Major drawbacks of dynamic analysis are the difficulty of instrumenting an existing application and the potential perturbation of application execution by the instrumentation. Techniques such as sampling or profiling reduce the cost of instrumentation. In sampling, from a limited set of application executions, a generalized model of application behavior is extrapolated. Sampling is only statistically accurate. In profiling, an application is executed in a series of expected situations. Profiling requires that profile scenarios accurately represent the day-to-day usage of the application. A scenario a set of conditions and inputs under which an application is run. In the COIGN system, scenario-based profiling can be used to estimate an application's run-time

behavior.

Detailed Description Text - DETX (91):
Late binding of an application across a specific network is facilitated by two mechanisms, described in detail below. First, compression of information about application communication reduces ADPS run-time overhead during profiling, and thereby enables more accurate and efficient summarization of network-independent communication costs. Second, quick estimation of the latency and bandwidth of a network allows the ADPS to delay partitioning until current estimates are needed. Combined, these techniques make it possible to delay binding of a distribution to a network until the latest possible moment, thus facilitating automatic adaptation to new networks.

Detailed Description Text - DETX (139):
The illustrated ADPS can identify application units for distribution according to a dynamic classification scheme. The word "dynamic," as it is used here, refers to classification incorporating information on how the application unit was used during run-time.

Detailed Description Text - DETX (147):
In the illustrated ADPS, a procedure-call-chain (PCC) classifier 264 can be used for dynamic classification. In the field of dynamic memory allocation, PCCs have been used to identify allocation sites for storing objects in memory. The PCC classifier 264 creates a classification descriptor by concatenating the static type of the component with the PCC of the instantiation request. A PCC consists of the return address from each of the invocation frames in the call stack. A depth-n PCC is a PCC containing the return addresses from the topmost n invocation frames. The depth of the PCC can be tuned to evaluate implementation tradeoffs. Accuracy in predicting allocation lifetimes increases as the depth of a PCC increases. While a PCC can be adequate for dynamic classification in procedure-based application, component-based applications have more call context because they are inherently object-oriented. The possible PCCs form a sparse, one-dimensional space: the range of valid return addresses. Object-oriented programming adds a second dimension: the identity of the component executing the code.

Detailed Description Text - DETX (154):
Generally, there are three classes of solutions to accomplish this task according to the present invention: modify the application's source code, modify the application's binaries prior to execution, or manipulate the application's execution through run-time intervention. Static modification of application source code or binaries is extremely difficult because it requires problematic whole-program static analysis. Manipulating the application's execution through run-time intervention is relatively straightforward but has some limitations. In general, an application's execution can be manipulated to produce a chosen distribution efficiently by intercepting calls to unit activation functions and executing them on the appropriate remote host.

Detailed Description Text - DETX (197):
A first version of interface informer is included in the heavyweight instrumentation package and operates during scenario-based profiling. This "profiling" interface informer uses format strings generated by the MIDL compiler and interface marshaling code to analyze all function call parameters and precisely measure inter-component communication. The profiling interface informer adds a significant amount of overhead to execution run-time.

Detailed Description Text - DETX (217):
To understand component behavior, COIGN gathers intimate knowledge of how an application and its components interact with the COM run-time services. COIGN is a binary-level system. The COIGN runtime penetrates the boundary between the application and the COM runtime transparently to the application. COIGN inserts itself between the application and the COM runtime services.

Detailed Description Text - DETX (229):
Although inline indirection is complicated somewhat by the variable-length instruction set of certain processors upon which the COIGN system runs, for example, the Intel x86 architecture, its low run-time cost and versatility more than offset the development penalty. Inline redirection of the CoCreateInstance function, for example, creates overhead that is more than an order of magnitude smaller than the penalty for breakpoint trapping. Moreover, unlike DLL redirection, inline redirection correctly intercepts both statically and dynamically bound invocations. Finally, inline redirection is much more flexible than DLL redirection or application code modification. Inline redirection of any API function can be selectively enabled for each process individually at load time based on the needs of the instrumentation.

Detailed Description Text - DETX (230):

To apply inline redirection, the COIGN runtime, a collection of DLLs, is loaded into the application's address space before the application executes. One of these DLLs, the COIGN run-time executive (RTE), inserts the inline redirection code.

Detailed Description Text - DETX (233):

Using one of several mechanisms, the COIGN runtime is loaded into the application's address space before the application executes. The COIGN runtime is packaged as a collection of dynamic link libraries. The COIGN run-time executive (RTE) is the most important DLL; it loads all other COIGN DLLs, so is loaded first into the application's address space. The COIGN RTE can be loaded by static or dynamic binding with the application.

PALM INTRANET

Day : Saturday
Date: 1/8/2005
Time: 17:04:29

Inventor Name Search Result

Your Search was:

Last Name = TIP
First Name = FRANK

Application#	Patent#	Status	Date Filed	Title	Inventor Name 19
60418643	Not Issued	159	10/15/2002	PROCESS FOR CONCENTRATING PRECIOUS METALS FROM SO-CALLED "COMPLEX" SOURCES, SUCH AS SAND, ROCK AND SLAG, BY A COMBINATION OF: MECHANICS, "DRI", METALLURGY, PHASE TECHNOLOGY, ELECTROLYSIS, AND CHEMISTRY	TIPPETT, FRANK D.
60233591	Not Issued	159	09/18/2000	SCALABLE PROPAGATION-BASED CALL GRAPH CONSTRUCTION ALGORITHMS	TIP, FRANK
60086850	Not Issued	159	05/27/1998	METHOD AND APPARATUS FOR PROCESSING IRON ORE TO ELECTROLYTIC IRON AND MARKETABLE METALLIC FILTRATE	TIPPETT , FRANK D.
60025285	Not Issued	159	09/20/1996	METHOD AND APPARATUS FOR A MULTI-PURPOSE, SHOCK WAVE THERMAL DUST REACTOR	TIPPETT , FRANK D.
60019183	Not Issued	159	06/05/1996	METHOD AND APPARATUS FOR RECYCLING IRON AND ZINC RICH WASTES	TIPPETT , FRANK D.
60017831	Not Issued	159	06/06/1996	METHOD AND APPARATUS FOR GENERATING ELECTRIC POWER	TIPPETT , FRANK D.
10960203	Not Issued	020	10/07/2004	PARAMETERIZATION OF PROGRAMMING STRUCTURES	TIP, FRANK
10673857	Not Issued	030	09/29/2003	AUTOMATIC CUSTOMIZATION OF CLASSES	TIP, FRANK
10153055	Not Issued	030	05/21/2002	SEMANTICS-BASED COMPOSITION OF CLASS HIERARCHIES	TIP, FRANK
09823060	Not Issued	071	03/30/2001	SCALABLE PROPAGATION-BASED METHODS FOR CALL GRAPH CONSTRUCTION	TIP, FRANK
09408224	6546551	150	09/28/1999	METHOD FOR ACCURATELY EXTRACTING LIBRARY-BASED OBJECT-ORIENTED APPLICATIONS	TIP , FRANK
09287818	6301700	150	04/07/1999	METHOD AND APPARATUS FOR SLICING CLASS HIERARCHIES	TIP , FRANK

09211177	6463581	150	12/14/1998	METHOD FOR DETERMINING REACHABLE METHODS IN OBJECT-ORIENTED APPLICATIONS THAT USE CLASS LIBRARIES	TIP , FRANK
09211176	6654951	150	12/14/1998	REMOVAL OF UNREACHABLE METHODS IN OBJECT-ORIENTED APPLICATIONS BASED ON PROGRAM INTERFACE ANALYSIS	TIP , FRANK
09159951	6279149	150	09/24/1998	AGGREGATE STRUCTURE IDENTIFICATION AND ITS APPLICATION TO PROGRAM ANALYSIS	TIP , FRANK
08942520	6230314	150	10/02/1997	OBJECT ORIENTED DEVELOPMENT TOOL MAKE ATTRIBUTES AND METHODS FROM CLASSES INDIVIDUALLY SELECTABLE AND COPY ONLY THOSE SELECTED TO THE NEW CLASS USED IN THE PROGRAM	TIP , FRANK
08942519	5983020	250	10/02/1997	RULE-BASED ENGINE FOR TRANSFORMATION OF CLASS HIERARCHY OF AN OBJECT-ORIENTED PROGRAM	TIP , FRANK
08794986	6179491	150	02/05/1997	METHOD AND APPARATUS FOR SLICING CLASS HIERARCHIES	TIP , FRANK
06629485	4587980	150	07/10/1984	PORTABLE CIGARETTE HOLDER, EXTINGUISHER AND ASHTRAY	TIPPER , FRANK

Inventor Search Completed: No Records to Display.

Search Another: Inventor **Last Name** TIP **First Name** FRANK Search

To go back use Back button on your browser toolbar.

Back to PALM | ASSIGNMENT | OASIS | Home page

PALM INTRANET

Day : Saturday
Date: 1/8/2005
Time: 17:05:39

Inventor Name Search Result

Your Search was:

Last Name = PALSBERG
First Name = JENS

Application#	Patent#	Status	Date Filed	Title	Inventor Name 3
60233591	Not Issued	159	09/18/2000	SCALABLE PROPAGATION-BASED CALL GRAPH CONSTRUCTION ALGORITHMS	PALSBERG, JENS
09823060	Not Issued	071	03/30/2001	SCALABLE PROPAGATION-BASED METHODS FOR CALL GRAPH CONSTRUCTION	PALSBERG, JENS
08620502	5946490	150	03/22/1996	AUTOMATA-THEORETIC APPROACH COMPILER FOR ADAPTIVE SOFTWARE	PALSBERG , JENS

Inventor Search Completed: No Records to Display.

Search Another: Inventor **Last Name** PALSBERG **First Name** JENS Search

To go back use Back button on your browser toolbar.

Back to PALM | ASSIGNMENT | OASIS | Home page